

# HARDWARE ACCELERATION OF AGENT-BASED SIMULATIONS OF SOCIAL MEDIA

Anna Gausen\*, Ce Guo\*, Wayne Luk\*

\* Imperial College London

London, SW7 2AZ, United Kingdom

{anna.gausen16, c.guo, w.luk}@imperial.ac.uk

## KEYWORDS

Probabilistic Model, Parallel Methods, Social Science, Policy-Making, Decision-Making

## ABSTRACT

Using agent-based simulations to model and understand social media platforms is a growing area of research. However, these models need to simulate more realistic population sizes and inter-agent connections to accurately capture the complex dynamics of the real system and be trusted to inform policy-making. This paper accelerates an agent-based simulation of X, previously known as Twitter. Accelerating agent-based simulations is challenging because the interactions between agents are unpredictable, the progress of simulated time must be respected, and the models have heterogeneous activity levels in time and space. The contributions of this paper are: (1) To develop a multi-core CPU implementation of an agent-based simulation of a social network, that can simulate inter-agent connections at the actual scale seen on X. (2) Port the function that controls agent actions and interactions in each timestep to GPU code using CUDA. (3) Benchmark the performance of the GPU and multi-core CPU implementations against a single-core CPU implementation. The GPU acceleration achieves up to a 180-fold speed-up compared to the single-threaded CPU implementation.

## INTRODUCTION

There is growing research into using agent-based simulations to model and understand social media platforms. The models treat each individual user as an agent in a simulated social network and can capture the realistic dynamics of information propagation through the network over time. This can help with understanding how information spreads on different platforms (Murić et al. (2022)), evaluating the effectiveness of misinformation countermeasures (Gausen et al. (2021)), and understanding the impact of changing recommendation objectives (Gausen et al. (2022)).

These models have significant potential for policy-making and evaluation of social media platforms, which is a growing area of importance (UK Government

(2023)). However, for these models to accurately capture the complex dynamics of the real system and be trusted for informing policies, they need to be able to model populations of agents and, importantly, inter-agent connections at a scale comparable to real social media platforms. However, agent-based simulations are computationally demanding (Fain and Dobrovolny (2022)) and require intensive memory access (Guo et al. (2020)).

The aim of this paper is to accelerate an agent-based simulation of a social media platform based on X, previously known as Twitter. The motivation behind this is twofold. Firstly, to decrease the time taken for a simulation to run. Faster runtimes will make more granular calibration processes feasible as these processes require the simulation to be run thousands of times. This will lead to model parameters that better fit the data. Secondly, to enable larger scale simulations both in terms of population size and inter-agent connections. Realistic sized inter-agent connections are extremely important when modelling social media platform as they are a driving force behind information propagation and the recommendation algorithm behaviour (Hesam et al. (2021)). This will unlock the potential of these simulations for policy-making and decision-making.

This paper describes acceleration of a model published by Gausen et al. (2022), which ran simulations of 1,000 agents with 100 connections. There are a number of challenges associated with accelerating this model:

1. The interactions between agents are dynamic and unpredictable leading to high data transfer overheads.
2. The progress of simulated time must be respected, which constrains parallel execution and limits parallelisation.
3. The model is heterogeneous in time and space, making distribution of the workload challenging.

We attempt to address these challenges with our accelerations. Firstly, we cap the maximum number of neighbors to prevent over-allocation of pre-allocated memory. Secondly, we fix neighbors' states and attributes at the beginning of each timestep to minimize memory allocation and enable full parallelization of agents

within a timestep. Thirdly, we implement both multi-threaded CPU and GPU acceleration to compare their performance with the uneven distribution of workload and random memory access.

In this paper we implement two hardware accelerations. Different types of hardware are suited to different tasks and overcome different challenges. Tasks that have unpredictable data access or are sequential in logic are more suited to CPUs, whilst tasks with very fine grained parallelism are better suited to GPUs, which have highly parallelised architectures (Xiao et al. (2019)). Therefore, we implement both a multi-core CPU acceleration and a GPU acceleration, then compare these to a benchmark of a single-core CPU implementation. The contributions of this paper include the following:

1. Develop a multi-core CPU implementation of an agent-based simulation of a social media platform, based on Gausen et al. (2022). This can simulate inter-agent connections at the actual scale seen on X.
2. Port the function that controls agent actions and interactions in each timestep to GPU code using CUDA, as this is the most computationally demanding part of the simulation.
3. Benchmark the performance of the GPU and multi-core CPU implementations against a single-core CPU implementation. The GPU acceleration achieves a highest speed up of approximately 180 times, compared to this benchmark.

## BACKGROUND AND RELATED WORK

In this paper we will consider parallel computing in relation to agent-based simulation. Agent-based simulations can be applied to many domains including epidemiology (Li and Giabbanelli (2021)), transport (Jing et al. (2020)), and social networks (Gausen et al. (2021)). The main characteristics of these simulations is that they are made up of populations of autonomous agents that have simplistic behaviours on the micro-scale but when they interact more complex phenomena emerge at the macro-scale, also known as emergent behaviour.

Agent-based simulations are computationally demanding and well placed for parallel computing techniques. They are structured well for parallelisation: they contain many independent agents and they are often timestepped so that agents update within each timestep, providing independence within each timestep and the potential for parallel execution (Xiao et al. (2019)). There is existing research on accelerating agent-based simulations with hardware. However, the general frameworks, such as FLAME GPU (Richmond et al. (2023)) and MASS CUDA (Kosiachenko et al. (2019)), are not designed to support simulating a recommendation algorithm that mediates inter-agent communication. Based

on our review, there are no existing hardware accelerations of agent-based simulations of social media platforms that include the recommendation algorithm. This is a crucial aspect of the system when modelling social media platforms like X.

There are a number of challenges for acceleration of agent-based simulation due to their characteristics (Xiao et al. (2019)): (1) There is a high interactivity. Due to the agents' interactivity there is often frequent data transfers among hardware devices to capture inter-agent communication resulting in large overheads. This interaction is usually dynamic and unpredictable, leading to random memory access. This is challenging for accelerators that prefer regular, linear memory access. (2) Simulated time is a central feature of agent-based simulation. This can limit parallelism as it puts constraints on the parallel execution in order to maintain the progress of the simulation in a way that respects the synchronisation of agent data at each timestep. (3) There is a high degree of heterogeneity within the models (Crooks and Heppenstall (2011)). The agents act and interact based on their individual attributes, internal states, neighbours' attributes, and the current environment. This means that there is heterogeneity in time and space. At some points in simulated time there may be regions of the network with little activity and therefore low computational demands, whilst other regions are highly active. This results in the hardware challenge of dividing the simulation workload across processing elements.

In this paper we will focus on two approaches to parallel computing: Multi-Core CPUs and GPUs. Multi-core CPUs are adept at handling complex logic and branching, such as complex intra-agent interaction rules. They are able to execute mostly unchanged parallelised C++ code. There are a number of agent-based simulation frameworks for parallelized execution on CPU-based environments e.g. Repast-HPC (Collier and North (2011)). In addition, CPUs are generally better suited to tasks with large amounts of random memory access than GPUs due to their cache architecture.

Graphical Processing Units (GPUs) have smaller cores with highly parallel architecture. For settings where large amounts of data can be processed in parallel, GPUs have an advantage over CPUs. In the case of agent-based simulation, they are effective if agent rules are uniform, interactions are local, and agent actions can be parallelised. There are frameworks and libraries, such as CUDA (Nvidia Corporation (2024)), OpenCL, and Thrust (Bell and Hoberock (2012)), that make development simpler than Field Programmable Gate Arrays (FPGAs) (Escobar et al. (2015)). However, the programming model is more restrictive than CPU and it requires a deeper understanding of GPU architecture due to the programming model and the complexity in hardware configuration (Wąs et al. (2015)). One of the biggest challenges is memory intensive tasks and

Table 1: Table showing the percentage of time taken for each process within the simulation where (A) Graph Initialization; (B) Agent Initialization; (C) Agent Step Function; (D) Post-processing. These results were generated with the single core CPU implementation with population = 50,000, average agent connections = 700 and timesteps = 100.

Process	A	B	C	D
Time (%)	11.58	0.35	86.53	1.54

tasks with complex data dependencies. Memory management between CPU and GPU can introduce bottlenecks. In this paper, we accelerate the agent-based simulation with both a multi-core CPU implementation and a GPU-based implementation. These two approaches enables us to compare the performance between them under different run configurations.

## IMPLEMENTATION

In this section, we present details of our implementation, including descriptions of the agent-based simulation and the two implemented accelerations.

### The Agent-based Simulation of a Social Media Platform

The agent-based simulation used in this paper is simulating a social network,  $X$ , and is based the model presented by Gausen et al. (2022). Here we will outline the core logic of the agent-based model in a sequential manner, as the parallelised implementations will be discussed in the following sections. In this model, the agents represent social media users and they are connected to other users in the network. A set of probabilities determine the users’ behaviour: (1) Probability that an agent is online in a given timestep  $P_{\text{online}}$ . (2) Probability that the agent retweets a tweet they view  $P_{\text{reshare}}$ . (3) Probability that an agent rejects a tweet they view and will not retweet it now or in the future  $P_{\text{reject}}$ . For more information on how the agent-based simulation operates, see Algorithm 1.

For this simulation there are three key problem-space parameters that significantly influence runtime: number of agents, average number of connections between agents, and number of simulated timesteps. The most compute-intensive operation is the simulation of the AgentStep function. For a population size of 50,000, agent connections of 700 and 100 timesteps, this takes approximately 86% of the total runtime, as shown in Table 1. In the AgentStep function, an agent interacts with other agents’ posts through viewing their simulated newsfeed. This requires the model to access data from all other agents that an agent is connected to (Hesam et al. (2021)). This action is parallelizable within a given

---

### Algorithm 1 Agent-Based Simulation Overview

---

- 1: Read in parameters: population, node degree, timesteps, probabilities
  - 2: Initialize graph
  - 3: Initialize agent attributes and states
  - 4: Simulated timesteps:
  - 5: **for** each timestep **do**
  - 6:     **for** each agent **do**
  - 7:         **procedure** AGENTSTEP(agent, t)
  - 8:             **if** agent is online **then**
  - 9:                 AgentPost?
  - 10:                 GenerateNewsfeed(neighbors,  $N_{\text{posts}}$ )
  - 11:                 Agent views top  $N_{\text{posts}}$  posts
  - 12:                 AgentRetweet?
  - 13: Post-processing simulation results
- 

timestep, assuming that the agent only needs information about neighboring agents based on their state at the start of each timestep. This assumption is realistic when a timestep represents a small unit of time. This paper aims to accelerate this operation using parallel computing hardware, specifically through implementations on both multi-core CPUs and GPUs.

### Multi-Core CPU Implementation

The first acceleration that was implemented was a multi-core CPU implementation. In this case, we applied multi-core parallelisation across all agents within each timestep. Reflecting on Algorithm 1, for each timestep (line 5) the agents are distributed across  $N_{\text{threads}}$  threads on the CPU. In the CPU implementation, for each timestep, the threads are created. Then a number of agents are allocated to each thread. The threads run in parallel. In each thread, each agent is run in sequence. For each agent, the AgentStep function is run which simulates the agent’s actions in that timestep, including viewing their simulated newsfeed or retweeting. Figure 1 outlines the thread allocation and logic for this multi-core CPU implementation.

### GPU Implementation

The second acceleration that was implemented was a GPU implementation. For this implementation, we ported the parallelisation across all agents within each timestep to GPU using CUDA. CUDA is the NVIDIA developed general-purpose parallel computing architecture (Falk et al. (2011)). A novel aspect of this implementation is the GPU function that models the recommendation algorithm behaviour in each timestep. Previous papers model agent communication (Richmond et al. (2023)), but they cannot model inter-agent communication that is mediated through a recommendation algorithm. Figure 2 illustrates the implementation, including the logic of each thread and how these threads

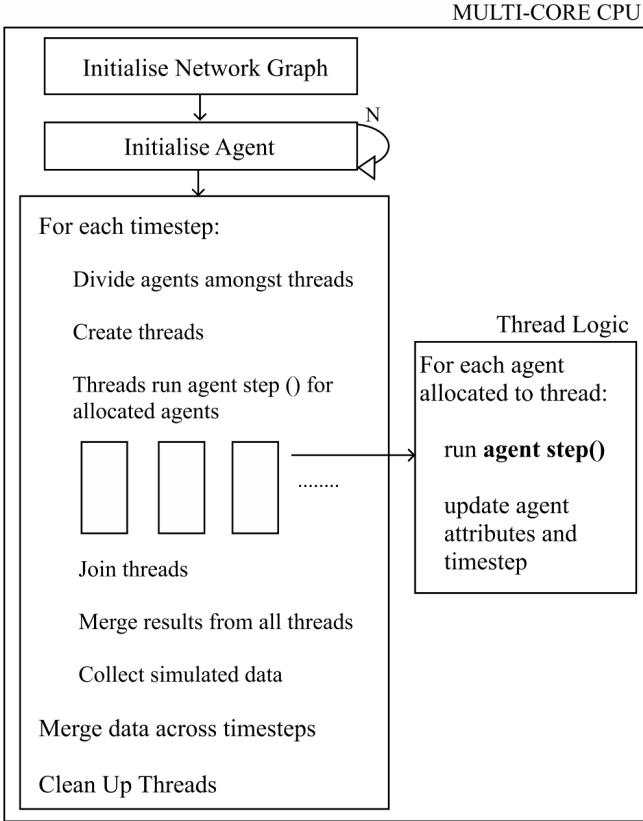


Figure 1: Figure showing the Multi-Core CPU implementation including thread allocation and logic.

are mapped to hardware units (Host/Device). After the graph and agent initialization occur on the CPU (Host), the memory allocation and initialization for GPU resources are completed before the timestep loop. The maximum size of a thread block ( $d$ ) is 1024 (Falk et al. (2011)). The kernel configuration for this paper was set as number of threads per a block  $d = 512$  and the number of blocks per a grid ( $B$ ) is set to dynamically change with population size  $N_{pop}$ , see Equation (1). This is necessary because each agent requires a corresponding thread in order to be processed.

$$B = (N_{pop} + d - 1)/d; \quad (1)$$

Within the timestep loop, the CUDA kernel is launched. On the GPU, a thread is allocated to each agent to carry out the sequential logic within the AgentStep function. After launching the kernel for each timestep, the CPU waits for the GPU to finish executing the kernel. Results from the GPU tracking the information propagation and states of each agent within a timestep are copied back to the host (CPU) memory after each timestep. Once all timesteps have completed, the GPU is cleaned up and the post-processing occurs on the host.

There are some specific challenges when implementing agent-based simulations on GPUs due to their architecture and memory limitations, as outlined in previ-

ous sections. Firstly, the interactions between agents are dynamic and unpredictable. This means that the simulation requires frequent, dynamic memory allocation, which can lead to bottlenecks and high data transfer overheads. In addition, getting information from neighboring agents can be inefficient on GPUs because neighbours may be located anywhere in the global memory. In the case of social media, the number of inter-agent connections can be high. On X, users have over 700 followers on average (Aslam (2024)). Secondly, the progress of simulated time usually must be respected, as with this implementation. This constrains parallel execution and limits parallelisation. Thirdly, the simulation can be heterogeneous in time and space, as with this implementation. In addition, within a kernel, agents can have differing rules of behaviour with different execution paths. This can lead to thread divergence. These challenges require careful consideration.

In our implementation we made a number of design decisions to approach these challenges. We make the assumption that within a timestep neighbors' states and attributes are fixed at the values at the start of the timestep. This assumption holds true for small timesteps and has two benefits. It minimises the frequency of memory allocation and enables agents to be run independently on the kernel, so it can be fully parallelised within a timestep. We limit the maximum number of neighbors to 1,000, to not over allocate pre-allocated memory for neighbouring agents and providing an upper limit on the search process. This number was chosen as only 0.06% of X users have a followership over 1,000 on X (Lucas (2024)).

## EVALUATION

This section will cover the empirical evaluation, including the benchmarking of results, experimental set up, the results of the CPU implementation, and the results of the GPU implementation.

### Benchmarking

The multi-core CPU and GPU results are benchmarked against a single core CPU implementation ( $N_{threads} = 1$ ). The performance is benchmarked using time, which is measured the same way in the benchmark, the multi-core CPU, and the GPU implementations, using the standard C++ toolbox. The time compared is the simulation time only. This does not include the pre-processing, such as graph initialisation, or the post processing, as these are independent of a simulation run and can be carried out in many different ways.

We are accelerating a specific model, so it is not possible to use existing published hardware accelerations as a benchmark. However, the results can be compared to the runtime of the non-accelerated model, developed using the MESA framework, published by Gausen et al.

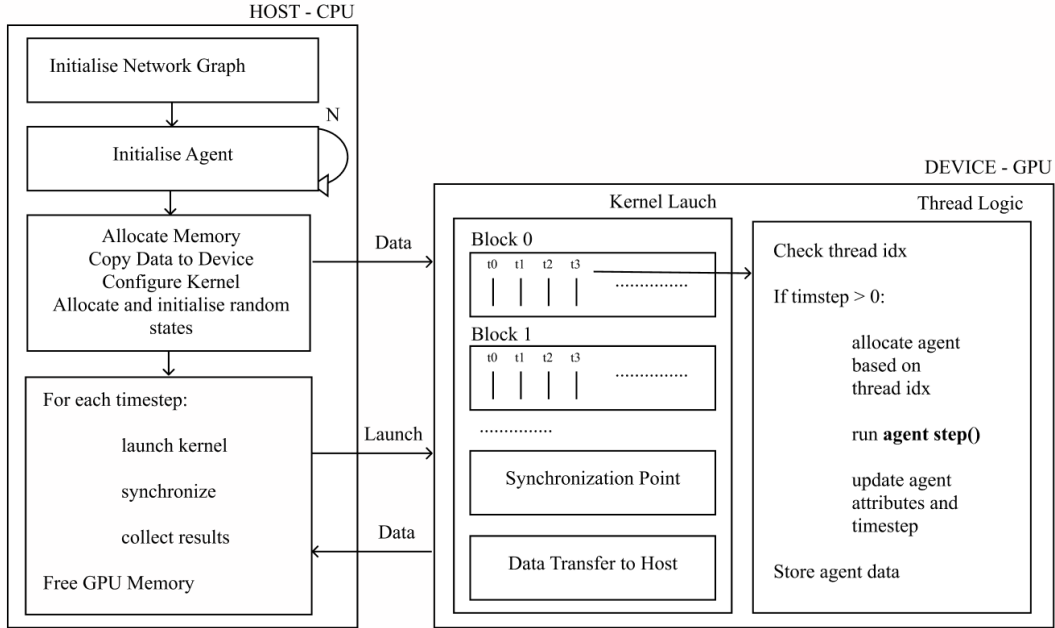


Figure 2: Figure showing GPU implementation, including the logic of each thread and how these threads are mapped to hardware units (Host/Device).

(2022).

In order to reproduce the simulation on different hardware platforms, we used the same random seed in the simulations and compared the results. As the random number generators vary on CPU and GPU, we manually defined a set of random numbers for the simulation and compared the outputs of the macro-level information propagation. In addition, we directly followed the interactions for specific agents. We then varied the set of random numbers to ensure the behaviour was the same under different conditions.

## Experimental Set-Up

The experiments were run with population size varying from 10,000 to 50,000 for the multi-core CPU and GPU implementation. This enables performance to be measured as population increases. For all experiments, the node degree value was fixed with an average of 700 (Aslam (2024)) and a maximum of 1,000 (Lucas (2024)). These values represent the actual scale of follower/followee numbers seen on the real platform being modelled, X.

Two servers with NVIDIA graphics cards are used in the evaluation:

- Server A comes with a 13th Gen Intel Core i3-13100F 4-core processor (8 threads, 4.5 GHz max frequency), with 32 GB of DDR4 memory. This server is equipped with a NVIDIA GeForce RTX 4090 graphics card with 24 GB of GDDR6X memory. Compilation were done with g++ (release Ubuntu 11.4.0-1ubuntu1 22.04) and CUDA compi-

lation tools (release 12,4, V12.4.99).

- Server B is equipped with an AMD Ryzen 7 PRO 3700 8-core processor (16 threads, 8 cores 4.43 GHz max frequency), with 32 GB of DDR4 memory. This server has a GeForce GTX 1660 SUPER graphics card with 6 GB of GDDR6 memory. Compilation were done with g++ (release Ubuntu 11.4.0-1ubuntu1 22.04) and CUDA compilation tools (release 12,4, V12.4.131).

Given that the AMD Ryzen 7 PRO 3700 processor on Server B offers significantly higher multi-core performance compared to the Intel Core i3-13100F processor on Server A, we use only Server B to evaluate the multi-core CPU implementation.

## Multi-Core CPU Results

The non-accelerated model published by Gausen et al. (2022) has a runtime of 19.09s for a population size of 1,000 and an average node degree of 100. The single-thread CPU runtime with the same parameters is 0.074s, achieving a 259-fold speed-up. In addition, the population size and node degree improvements of 50 times and 7 times, respectively, show a significant advancement. Particularly node degree, as the accelerated model can simulate inter-agent connections at the actual scale seen on X.

Table 2 provides the actual simulation runtime values in milliseconds for both the CPU and GPU implementation. The results in Table 2 show that as the population size increases, the runtime increases proportionally.

Table 2: Table showing how the simulation runtime changes with population size and number of CPU threads (N), for the CPU (server B) and GPU implementation (server A).

Population	N	CPU (ms)	GPU (ms)
10000	1	17908.30	252.10
20000	1	35600.10	308.96
30000	1	53870.50	371.52
40000	1	71733.70	433.30
50000	1	88815.30	488.45
10000	2	9552.62	252.10
20000	2	18659.30	308.96
30000	2	28510.70	371.52
40000	2	37736.60	433.30
50000	2	47049.90	488.45
10000	4	5467.70	252.10
20000	4	10604.50	308.96
30000	4	15785.90	371.52
40000	4	20969.00	433.30
50000	4	26504.10	488.45
10000	8	3373.82	252.10
20000	8	6625.96	308.96
30000	8	9834.58	371.52
40000	8	13128.80	433.30
50000	8	16457.80	488.45

This highlights how sensitive the simulation runtime is to the number of agents.

Figure 3 presents how the performance, in terms of runtime, of the CPU implementation changes with population size and number of threads. The x-axis shows the population size, which ranges from 10,000 to 50,000, and the y-axis shows the speed-up, which is speed increase compared to the single-thread CPU run for each population size and number of threads. Figure 3 shows that the simulation runtime decreases as the number of threads increases in the multi-core implementations, for a given population size. Increasing the number of threads from 1 to 8 results in above a 5-fold speed-up. Whilst this is an improvement, it still motivates the need for further acceleration through the GPU implementation.

## GPU Results

Figure 4 presents how the performance, in terms of runtime, of the GPU implementation changes with population size. The x-axis shows the number of agents simulated, which ranges from 5,000 to 50,000, and the y-axis shows the normalised performance, which is the run time normalised by the runtime for a population size of 5,000. The figure shows that as the population size increases, the runtime increases linearly. The one exception to this is when the population is its lowest level, at 5,000. This indicates that at low populations, where the runtime is lower, setting up the device may be a bigger proportion of the overall runtime and, therefore,

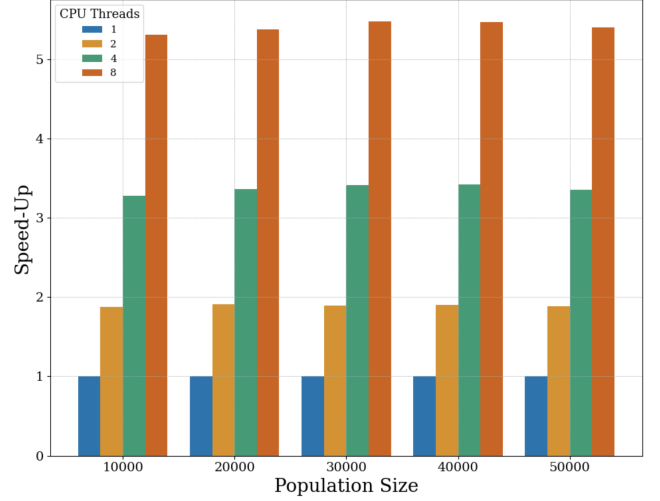


Figure 3: Figure showing how the runtime changes with population size and number of threads, for the CPU Implementation.

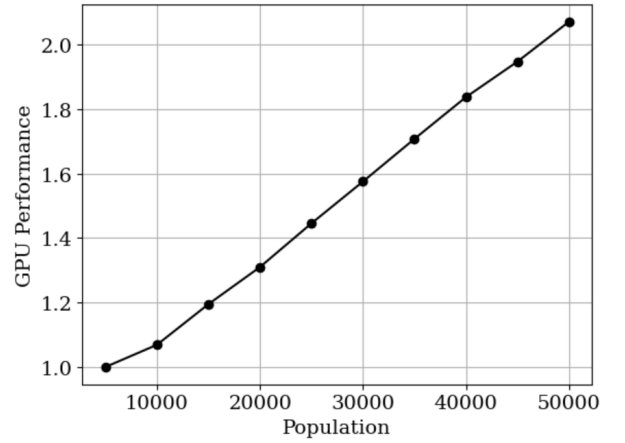
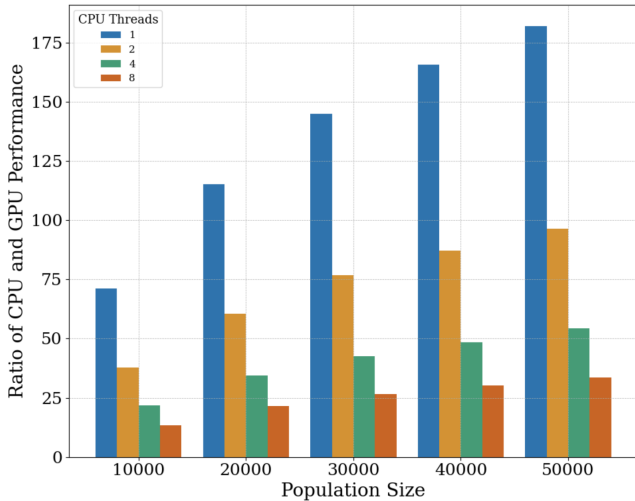


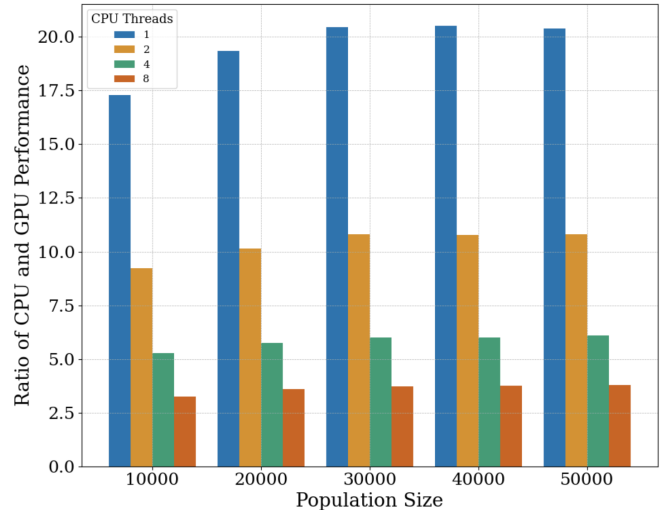
Figure 4: Figure showing how the runtime changes with population size, for the GPU implementation on server A. The y-axis shows the normalised performance, which is the run time normalised by the runtime for a population size of 5,000

dominate.

Figure 5 compares the performance of the GPU implementation to the multi-threads CPU implementation, for different numbers of threads and population sizes. The results show that the GPU implementation running on the GeForce RTX 4090 achieves up to a 180-fold speed-up compared to the single-threaded CPU implementation, marking a significant improvement. Additionally, the GPU implementation on the GeForce GTX 1660 SUPER achieves up to a 20-fold speed-up compared to the single-threaded CPU implementation. As the number of CPU threads increases, the speed-up reduces. However, for a population size of 50,000, the GPU performance is still over 30 times higher than the 8-thread CPU implementation for the GeForce RTX 4090. The figure also



(a) NVIDIA GeForce RTX 4090



(b) NVIDIA GeForce GTX 1660 SUPER

Figure 5: Speed-up of the GPU implementation compared to the multi-threaded CPU implementation

shows that as the population size increases, the speed-up of the GPU implementation increases. This is due to the enhanced parallelization offered by GPUs, which can be leveraged as the population grows, allowing the code to be parallelized across the population in each timestep.

In terms of the trend of the ratio between the GPU and CPU performance, as the population increases the performance ratio increases but is also plateauing. This indicates that there will be a maximum performance ratio that can be achieved by the GPU implementation. This performance ratio plateau will depend on the experimental set-up. On Server B, this plateau was reached at a population size of 30,000.

## CONCLUSION AND FUTURE WORK

The aim of this paper is to accelerate an agent-based simulation of a social media platform, based on Gausen et al. (2022). This is motivated by the needed to simulate more realistic populations of agents and numbers of inter-agents connections. This will help unlock the potential of agent-based simulations for policy-making and decision-making around social media platforms.

This paper presents two approaches to acceleration. The single core CPU implementation enables the model to be run for 50 times larger population sizes and achieves a 256-fold speed up compared to the previously published non-accelerated implementation (Gausen et al. (2022)). The accelerated model is also able to simulate realistic scaled follower/followee relationships based on X. The multi-core CPU implementation achieves over 5-fold speed-up when comparing 8 threads to the single-core CPU benchmark. The GPU implementation provides further acceleration. This achieves up to a 180-fold speed-up, compared to the single-core CPU bench-

mark on Server A. These results present considerable advancement both in the scale of the simulations and their performance.

There are a number of directions for future work. Firstly, the accelerated simulation should be run within a full pipeline which includes calibration to estimate the time savings of the speed-up. Secondly, the agent-based simulation could be accelerated with different hardware, such as FPGAs, for performance comparison. Thirdly, the agent-based simulation is modelling the social media platform X. Future work could extend this model to generalise to other platforms.

## ACKNOWLEDGEMENTS

Anna Gausen is supported by a studentship from the UKRI CDT in Safe and Trusted Artificial Intelligence (EP/S023356/1).

## REFERENCES

- Aslam S., 2024. *Twitter by the Numbers*. URL <https://www.omnicoreagency.com/twitter-statistics/>.
- Bell N. and Hoberock J., 2012. *Thrust: A productivity-oriented library for CUDA*. In *GPU computing gems Jade edition*, Elsevier. 359–371.
- Collier N. and North M., 2011. *Repast HPC: A Platform for Large-Scale Agent-Based Modeling*. In *Large-Scale Computing*. Wiley Online Library, 81–109.
- Crooks A.T. and Heppenstall A.J., 2011. *Introduction to agent-based modelling*. In *Agent-based models of geographical systems*, Springer. 85–105.

- Escobar F.A.; Chang X.; and Valderrama C., 2015. *Suitability analysis of FPGAs for heterogeneous platforms in HPC*. In *IEEE Transactions on Parallel and Distributed Systems*. IEEE, vol. 27, 600–612.
- Fain B.G. and Dobrovolny H.M., 2022. *GPU acceleration and data fitting: agent-based models of viral infections can now be parameterized in hours*. In *Journal of computational science*. Elsevier, vol. 61, 101662.
- Falk M.; Ott M.; Ertl T.; Klann M.; and Koepl H., 2011. *Parallelized agent-based simulation on CPU and graphics hardware for spatial and stochastic models in biology*. In *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*. Association for Computing Machinery, New York, NY, USA, CMSB '11, 73–82.
- Gausen A.; Luk W.; and Guo C., 2021. *Can We Stop Fake News? Using Agent-Based Modelling to Evaluate Countermeasures for Misinformation on Social Media*. In *Workshop Proceedings of International AAAI Conference on Web and Social Media (ICWSM)*. 1–8.
- Gausen A.; Luk W.; and Guo C., 2022. *Using Agent-Based Modelling to Evaluate the Impact of Algorithmic Curation on Social Media*. In *ACM Journal of Data and Information Quality: Special Issue on Trust and Truth Online*. Association for Computing Machinery, New York, NY, USA, vol. 15, 1–24.
- Guo C.; Luk W.; and Weston S., 2020. *Accelerating Simulation for Agent-based Epidemic Models using FPGAs*. In *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 1–8.
- Hesam A.; Breitwieser L.; Rademakers F.; and Al-Ars Z., 2021. *Gpu acceleration of 3d agent-based biological simulations*. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 210–217.
- Jing P.; Hu H.; Zhan F.; Chen Y.; and Shi Y., 2020. *Agent-based simulation of autonomous vehicles: A systematic literature review*. In *IEEE Access*. IEEE, vol. 8, 79089–79103.
- Kosiachenko L.; Hart N.; and Fukuda M., 2019. *MASS CUDA: a general GPU parallelization framework for agent-based models*. In *Advances in Practical Applications of Survivable Agents and Multi-Agent Systems: The PAAMS Collection: 17th International Conference, PAAMS 2019, Ávila, Spain, June 26–28, 2019, Proceedings 17*. Springer, 139–152.
- Li J. and Giabbanelli P., 2021. *Returning to a normal life via COVID-19 vaccines in the United States: A large-scale agent-based simulation study*. In *JMIR Medical Informatics*. JMIR Publications Toronto, Canada, vol. 9, 27419.
- Lucas M., 2024. *Understanding Twitter’s User Base*. URL [www.themarketingheaven.com](http://www.themarketingheaven.com).
- Murić G.; Tregubov A.; Blythe J.; Abeliuk A.; Choudhary D.; Lerman K.; and Ferrara E., 2022. *Large-scale agent-based simulations of online social networks*. In *Autonomous Agent Multi-Agent Systems*. Springer, Berlin, Germany, vol. 36, 38.
- Nvidia Corporation, 2024. *CUDA Toolkit*. URL <https://developer.nvidia.com/cuda-toolkit?ref=blog.mergify.com>.
- Richmond P.; Chisholm R.; Heywood P.; Chimeh M.K.; and Leach M., 2023. *FLAME GPU 2: A framework for flexible and performant agent based simulation on GPUs. Software: Practice and Experience*, 53, no. 8, 1659–1680.
- UK Government, 2023. *Online Safety Bill (Department for Science, Innovation and Technology and Department for Digital, Culture, Media & Sport)*.
- Waś J.; Mróz H.; and Topa P., 2015. *GPGPU computing for microscopic simulations of crowd dynamics*. In *Computing and Informatics*. vol. 34, 1418–1434.
- Xiao J.; Andelfinger P.; Eckhoff D.; Cai W.; and Knoll A., 2019. *A Survey on Agent-based Simulation Using Hardware Accelerators*. In *ACM Comput. Surv.* Association for Computing Machinery, New York, NY, USA, vol. 51, 1–35.

## AUTHOR BIOGRAPHIES

**Anna Gausen** is a PhD researcher studying Safe and Trusted Artificial Intelligence at Imperial College London. Her research focuses on agent-based modelling and recommendation algorithms.

**Ce Guo** is a research associate in accelerator computing at Imperial College London. His research focuses on efficient computing systems for large-scale temporal data analytics, agent-based modelling, and causal structural learning.

**Wayne Luk** is a professor at Imperial College London. He leads multiple research groups, including the Custom Computing Research Group, the EPSRC Centre for Doctoral Training in High-performance Embedded and Distributed Systems, and the Centre for Advanced Financial Engineering.